AIAA-97-2029

Multiple Semi-Coarsened Multigrid for 3D CFD

B. Koren*, P.W. Hemker, C.T.H. Everaars
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

A significant difficulty of standard multigrid methods for 3D problems, when compared to application to 2D problems, is that the requirements to be imposed on the smoother are much more severe. As a remedy, we investigate three different possibilities of multiple semi-coarsening: full, sparse and semi-sparse. Numerical results are presented for a standard 3D transonic test case. The good parallel computing properties of the sparse-grid and the semi-sparse-grid approaches are also investigated. The first speed-up results are promising. The paper contributes to the state-of-the-art in efficiently solving 3D fluid-flow equations.

1 Introduction

With standard multigrid methods, the total amount of work on the coarse grids is relatively smaller in the 3D case than in the 2D case. However, the reverse side is that in 3D only a relatively small amount of error components can be annihilated by the coarse-grid corrections. When cells are used as grid elements, in 3D, standard coarsening implies restriction from each set of $2 \times 2 \times 2$ cells to a single cell only. Because the set of eight cells can support more high-frequency errors than the two-dimensional 2×2–set, 3D standard multigrid imposes stronger requirements on the smoother than 2D standard multigrid. Standard multigrid may not perform satisfactory for 3D generalizations of 2D problems, for which it does perform well. A fix might be found in deriving a more powerful smoother, keeping the other components of the numerical method the same. A more natural remedy is not to apply standard, i.e. full coarsening, but to use multiple semi-coarsening instead (Figure 1). When multiple semi-coarsening is applied to solve a system of equations defined on the single, finest grid $\Omega_{l_{\max},m_{\max},n_{\max}}$, and when *all* coarser grids $\Omega_{l,m,n}$, level$\equiv l+m+n < l_{\max}+m_{\max}+n_{\max}$ contribute to the solution process, we speak of *full-grid-of-grids* semi-coarsening [12]. A disadvantage of full-grid-of-grids semi-coarsening is that many grid cells are needed in total. With N^3 the total number of cells on the finest grid $\Omega_{l_{\max},m_{\max},n_{\max}}$ in 3D, asymptotically standard multigrid uses $\frac{9}{8}N^3$ grid cells versus $8N^3$ cells for the full-grid-of-grids approach. An efficiency improvement can be achieved by thinning out the grid-of-grids, e.g. by deleting fine grids. This may lead to the sparse-grid-of-grids and the semi-sparse-grid-of-grids approaches, to be discussed in this paper.

The contents of the paper is as follows. In Section 2, we briefly describe the 3D discrete equations and the 3D test case to be considered throughout the paper. In Section 3, we describe the four different multigrid strategies to be compared: (i) standard multigrid, (ii) full-grid-of-grids multigrid, (iii) sparse-grid-of-grids multigrid and (iv) semi-sparse-grid-of-grids multigrid. Per type of multigrid strategy, we illustrate the performance for the test case chosen. Finally, in Section 4, we exploit the good parallelization properties of the sparse-grid and the semi-sparse-grid method, a speed-up analysis and speed-up results are presented.

2 Test set

2.1 Equations

The steady, non-isenthalpic, 3D Euler equations of gas dynamics are considered. The equations are discretized in their integral form. The computational domain Ω is divided, in a regular manner, into cell-centered finite volumes. These finite volumes are arbitrarily shaped hexahedra. Following the Godunov approach, along each cell face the flux vector is assumed to be constant and to be determined by a uniformly constant left and right state. To solve the resulting 1D Riemann problem, we apply the 3D extension of the 2D P-variant [8] of Osher's approximate Riemann solver [15]. For the left and right cell-face states, we take the first-order accurate approximations. At a later stage, these approximations can be replaced by higher-order accurate ones, in which case also limiters can be introduced. We emphasize that the major challenge is to know how to solve *first-order accurate* discretized, steady 3D Euler equations at ef-

*Member AIAA.

ficient, grid-independent convergence rates. Once this is known, solving higher-order accurate discrete, steady 3D Euler equations can be done by a standard procedure, e.g. by a defect correction method as outer and the efficient multigrid method as inner iteration [10, 11]).

2.2 Flow problem

As test case we consider the ONERA-M6 half-wing at $M_\infty = 0.84$, $\alpha = 3.06°$. The grids used are of C-O-type. (Graphs are given in [12].) The wing as well as the grids are symmetric with respect to the plane through the wing's leading and trailing edges. In the results to be presented hereafter, the finest grid considered is a $64 \times 16 \times 16$–grid.

3 Multigrid methods and results

3.1 Standard multigrid

3.1.1 Method

First we briefly describe the standard 3D multigrid algorithm. The multigrid methods to be described hereafter are based on it. We use the 3D generalization of the optimal 2D multigrid approach, that was originally described in [7, 8]. As the smoothing technique for the first-order discrete Euler equations, collective symmetric point Gauss-Seidel relaxation is applied. The four different symmetric relaxation sweeps that are possible on a regular 3D grid, are performed alternatingly. At each volume visited during a relaxation sweep, the system of five nonlinear equations is solved by Newton iteration.

As standard multigrid method we apply the nonlinear version (FAS, [3]), preceded by nested iteration (FMG, [3]). For this we construct a nested set of grids such that each finite volume on a coarse grid is the union of $2 \times 2 \times 2$ volumes on the next finer grid. Let $\Omega_0, \Omega_1, \ldots, \Omega_{\lambda_{\max}}$ be the sequence of such nested grids, with Ω_0 the coarsest and $\Omega_{\lambda_{\max}}$ the finest grid. Then, nested iteration is applied to obtain a good initial solution on $\Omega_{\lambda_{\max}}$, whereas nonlinear multigrid is applied to converge to the solution on the finest grid, $q_{\lambda_{\max}}$. The first iterand for the nonlinear multigrid cycling is the solution obtained by nested iteration. We proceed by discussing both stages in more detail.

Nested iteration The nested iteration starts with a user-defined initial estimate for q_0, the solution on the coarsest grid. To obtain an initial solution on a finer grid $\Omega_{\lambda+1}$, first the solution on the coarser grid Ω_λ is improved by a single nonlinear multigrid cycle. Hereafter, this solution is prolongated to the finer grid $\Omega_{\lambda+1}$. These steps are repeated until the highest level (finest grid $\Omega_{\lambda_{\max}}$) has been reached.

Nonlinear multigrid iteration Let $N_\lambda(q_\lambda) = 0$ denote the nonlinear system of first-order accurate discretized equations on Ω_λ, then a single nonlinear multigrid cycle is recursively defined by the following steps:

1. Improve on Ω_λ the latest obtained solution q_λ by application of n_{pre} relaxation sweeps.
2. Compute on the next coarser grid $\Omega_{\lambda-1}$ the right-hand side $r_{\lambda-1} = N_{\lambda-1}(q_{\lambda-1}) - I_\lambda^{\lambda-1} N_\lambda(q_\lambda)$, where $I_\lambda^{\lambda-1}$ is a restriction operator for right-hand sides.
3. Approximate the solution of $N_{\lambda-1}(q_{\lambda-1}) = r_{\lambda-1}$ by the application of n_{FAS} nonlinear multigrid cycles. Denote the approximation obtained as $\tilde{q}_{\lambda-1}$.
4. Correct the current solution by: $q_\lambda = q_\lambda + \tilde{I}_{\lambda-1}^{\lambda} (\tilde{q}_{\lambda-1} - q_{\lambda-1})$, where $\tilde{I}_{\lambda-1}^{\lambda}$ is a prolongation operator for solutions.
5. Improve again q_λ by application of n_{post} relaxations.

Steps (2),(3) and (4) form the coarse-grid correction. The restriction operator $I_\lambda^{\lambda-1}$ and the prolongation operator $\tilde{I}_{\lambda-1}^{\lambda}$ are the usual operators that are consistent with the piecewise constant approximation (for more details, see [12]).

3.1.2 Results

Convergence results are given in Figure 2. In both graphs, the residual ratio is defined as $\|R^i\|_{L_1} / \|R^1\|_{L_1}$, where R^i is the mass defect of the discrete Euler equations and where i refers to the status after the i-th iteration. For the standard multigrid convergence results presented in Figure 2b, we took $n_{\text{pre}} = 0$, $n_{\text{post}} = 1$, i.e. sawtooth-cycles. Though – of course – to a lesser extent than the single-grid convergence results (Figure 2a), the standard multigrid method's convergence results appear to be rather grid-dependent (Figure 2b). As mentioned in Section 1, the expected cure is to apply multiple semi-coarsening instead of standard, i.e. full coarsening. In the next sections we proceed by discussing this alternative coarsening.

3.2 Full-grid-of-grids multigrid

3.2.1 Method

Pioneering work has been done by Mulder [13], who has introduced multiple semi-coarsening as a fix to the poor convergence results observed in computing nearly grid-aligned flows governed by the

steady, 2D Euler equations. In [16], Radespiel and Swanson embroider on Mulder's approach for the steady, 2D Euler equations. Here we consider multiple semi-coarsened multigrid for the steady, 3D Euler equations, and pay particular attention to the prolongation operators.

Also in the case of the semi-coarsened multigrid method we use FAS as the basic multigrid algorithm, and on each grid collective symmetric point Gauss-Seidel relaxation is applied as the smoothing technique. As mentioned in Section 1, in the semi-coarsened multigrid method, the sequentially ordered set of grids $\Omega_\lambda, \lambda = 0, \ldots, \lambda_{\max}$, is replaced by a partially ordered set of grids $\Omega_{l,m,n}$, $l = 0, 1, \ldots, l_{\max}$, $m = 0, 1, \ldots, m_{\max}$, $n = 0, 1, \ldots, n_{\max}$, with $\Omega_{0,0,0}$ the coarsest and $\Omega_{l_{\max},m_{\max},n_{\max}}$ the finest grid. In the full-grid-of-grids variant of multiple semi-coarsening, all grids $\Omega_{l,m,n}$ play a role in the solution process. The nesting and the semi-coarsening relation between these grids and more data structure aspects are described in [9].

Nested iteration Also with semi-coarsening, nested iteration (FMG) is applied to obtain a good initial solution on the finest grid. We proceed to discuss the present nested iteration and nonlinear multigrid iteration in more detail. The nested iteration starts with a user-defined initial estimate on the coarsest grid, $\Omega_{0,0,0}$, which is improved by relaxation. To continue, the following two options exist:

- The approximate solution $q_{0,0,0}$ is prolongated to all grids up to and including level 3, with the 3D prolongation according to formula (37) in [6] (see [12] for the implementation in the present 3D Euler context). Next, the solution $q_{1,1,1}$ is improved by a single nonlinear multigrid cycle and prolongated to all grids up to and including level 6. For simplicity, we assume that $l_{\max} = m_{\max} = n_{\max}$. Then, the above process can be repeated in a straightforward manner up to and including level $3l_{\max}$. Notice that solution improvements are only made at $\Omega_{0,0,0}, \Omega_{1,1,1}, \Omega_{2,2,2}, \ldots$

- The approximate solution $q_{0,0,0}$ is prolongated to the three grids $\Omega_{1,0,0}$, $\Omega_{0,1,0}$ and $\Omega_{0,0,1}$ on the next level, with the same 3D prolongation as mentioned above. Next, the three solutions $q_{1,0,0}$, $q_{0,1,0}$ and $q_{0,0,1}$ are first improved by a single nonlinear multigrid cycle and then prolongated to all six grids on level 2. The above process is repeated up to and including level $l_{\max} + m_{\max} + n_{\max}$. Notice that here, as opposed to the previous strategy, solution improvements are made on *all* grids, *level-by-level.*

Nonlinear multigrid iteration A single nonlinear multigrid cycle on level $l + m + n$ is recursively defined by the following steps:

1. Compute the same right-hand sides as in standard multigrid, on all grids at the next coarser level $(l+m+n)-1$, but use as restriction operator the natural one described in [12] (natural because it just sums defects over the sub-cells).

2. Approximate the solutions on the coarser level $(l + m + n) - 1$ by the application of a single nonlinear multigrid cycle.

3. Correct the current solutions on level $l+m+n$ by one of two alternative correction prolongations. The first prolongation can be seen as an extension to 3D and to systems of equations, of the prolongation due to Naik and Van Rosendale [14]. It uses prolongation weights that are proportional to the absolute values of the restricted defect components. The second correction prolongation is the hierarchical one proposed in [6], equation (36). It has a-priori known prolongation weights $+1$ and -1.

4. Improve the solutions on level $l+m+n$ by the application of n_{post} relaxation sweeps.

3.2.2 Results

We first compare the two correction prolongations just mentioned: the one with defect-dependent weights and the one with fixed hierarchical weights. The nested iteration applied is the first one described in Section 3.2.1. Convergence results are shown in Figure 3. In the two graphs, the residual ratio is defined as $\|R^{i_{\text{FAS}}}\|_{L_1} / \|R^1\|_{L_1}$, where $R^{i_{\text{FAS}}}$ is the first component (i.e. the mass component) of $N_{l_{\max},m_{\max},n_{\max}}(q^{i_{\text{FAS}}}_{l_{\max},m_{\max},n_{\max}})$ and where i_{FAS} refers to the status after the i_{FAS}-th FAS-cycle. Similar as for the standard multigrid convergence results (Figure 2b), here we also used sawtooth cycles ($n_{\text{pre}} = 0$, $n_{\text{post}} = 1$). The improvement of both semi-coarsened multigrid methods with respect to the standard multigrid method is significant. Of both methods, the one with the fixed hierarchical prolongation weights (Figure 3b) performs better than the one with defect-dependent prolongation weights (Figure 3a).

The convergence results may still be further improved. In Figure 4 we present results for the same solution strategy as that of Figure 3b, but now with V-cycles ($n_{\text{pre}} = n_{\text{post}} = 1$) and with the more elaborate, level-by-level nested iteration described in Section 3.2.1.

3.3 Sparse-grid and semi-sparse-grid multigrid

3.3.1 Methods

As mentioned in Section 1, a disadvantage of full-grid-of-grids semi-coarsening is that $8N^3$ grid cells are needed in total (N^3 being the number of grid cells on $\Omega_{l_{\max},m_{\max},n_{\max}}$). An efficiency improvement can be achieved by deleting fine grids. Then, if no finest grid is available anymore, accurate approximations can still be constructed either by extrapolation or by the use of hierarchical bases. Most ambitious is the sparse-grid-of-grids approach. With the full grid-of-grids depicted as a cube in Figure 5a, the corresponding sparse grid-of-grids is the subset given in Figure 5c, only grids $\Omega_{l,m,n}$, level$\leq l_{\max}$ contribute. The reduction in the numbers of grid cells is enormous. The computational complexity of the sparse-grid-of-grids approach is $\mathcal{O}(N\log^2 N)$, i.e. almost the complexity of a 1D problem only! Theoretically, the sparse-grid-of-grids approach has the best ratio of discrete accuracy over number of grid points used [5], the loss of accuracy is only a logarithmic factor when compared with the full-grid-of-grids approach. In practice, although very fast, the accuracy of the sparse-grid approximations is slightly disappointing. It appears that more accurate approximations are obtained *not* by only increasing the number of levels, but also by dropping the cells with extreme aspect ratios. This leads to the compromise of the semi-sparse grid-of-grids. This uses the family of grids $\Omega_{l,m,n}$, level$\leq 2l_{\max}$, $\max(l,m,n) \leq l_{\max}$ (Figure 5b), which (asymptotically) still has a computational complexity which is smaller than that of the single-grid approach, viz. $\mathcal{O}(N^2\log N)$, i.e. almost the complexity of a 2D problem. So, though to a lesser extent than the genuine sparse-grid approach, it still is a cure to cubic complexity, the 'curse of 3D'.

3.3.2 Results

The numerical ingredients of both approaches are identical to those in the full-grid-of-grids approach applied in obtaining Figure 4. Exactly the same method is applied, with as the only difference that in the sparse-grid case the multi-level semi-coarsening solver stops its work at level $l_{\max}$. From there the solution is prolongated to the very finest grid at level $3l_{\max}$. The prolongation is done by the 3D extension of the combination extrapolation given on p. 290 in [17]. In the semi-sparse-grid approach the semi-coarsened multi-level algorithm is only stopped at level $2l_{\max}$ and from there, by the same combination technique, the finest-grid solution at $3l_{\max}$ is computed. A particular advantage of the semi-sparse-grid approach as compared to the sparse-grid approach, is that the 3D extrapolation rule can be applied for all remaining grids to be filled, including the grids along the boundaries of the grid-of-grids. In the sparse-grid approach this is not possible. There, for all boundary grids in between $l_{\max}$ and $2l_{\max}$ one has to make a compromise, for instance by applying a 2D or even a 1D combination extrapolation, which will inevitably result in some additional loss of accuracy.

In the Figures 6a–c we give an impression of the accuracy of the numerical solutions, obtained by the three different grid-of-grids approaches depicted in Figure 5. The reference solution is the fully converged $\mathcal{O}(h)$ finest grid solution (Figure 6a). It has been obtained on a $64\times16\times16$ C-O-type grid, it is the target for both solutions presented in Figures 6b and 6c. Of course, the semi-sparse grid solution (Figure 6b) comes closer to the reference solution. The sparse-grid solution (Figure 6c) is far off, but it has been obtained at extremely low computational cost as compared to both the semi-sparse-grid approach and the full grid-of-grids approach. In Table 1 we give the relative computing times used.

4 Parallelization of sparse-grid and semi-sparse-grid multigrid

4.1 Natural concurrency of both methods

The pre- and post-relaxations, steps 1 and 5 in the nonlinear multigrid iteration (Section 3.1.1), are done by a procedure (subroutine `scanlv`, see [4]) for performing a user-defined operation on all grids $\Omega_{l,m,n}$ at grid level $l+m+n$. In this case the user-defined procedure is the point Gauss-Seidel relaxation on all cells of grid $\Omega_{l,m,n}$. Because the relaxation subroutine only reads and writes data concerning its own grid, the relaxations can be done directly in parallel for all grids visited at a certain grid level. Given the fact that almost all computing time consumed by the total program, is used in the relaxations, parallel implementation is expected to pay off.

4.2 The Manifold coordination language

Parallelization is done through the MANIFOLD coordination language. MANIFOLD is a language for managing complex, dynamically changing interconnections among sets of independent, concurrent, cooperating processes [1]. MANIFOLD is based on the IWIM model of communication [2].

The basic concepts in the IWIM model are *processes*, *events*, *ports* and *channels* [4].

A MANIFOLD application consists of a (potentially very large) number of processes running on a network of heterogeneous hosts, some of which may be parallel systems. Processes in the same application may be written in different programming languages.

The MANIFOLD system consists of a compiler, a run-time system library, a number of utility programs, libraries of built-in and pre-defined processes, a link file generator called MLINK and a run-time configurator called CONFIG. The system has been ported to several different platforms (e.g. SGI 5.3, SUN 4, Solaris 5.2, and IBM SP/1). MLINK uses the object files produced by the (MANIFOLD and other language) compilers to produce link files needed to compose the executable files for each required platform. At the run time of an application, CONFIG determines the actual host(s), where the processes (created in the MANIFOLD application) will run.

The library routines that comprise the interface between MANIFOLD and processes written in other languages (e.g. C), automatically perform the necessary data format conversions when data are routed between various different machines.

4.3 Restructuring of the sequential 3D CFD code

The restructuring of the original sequential code can be described in a kind of master/slave protocol. In a coordinator process [4] we create and activate a master process that embodies the computations of the main program of the sequential version. When we arrive in the master process at a pre- or post-relaxation, the master delegates the relaxations to a separate slave process, for each single grid visited. Each time the master needs a slave, it raises an event to signal the coordinator to create the slave. In this way a pool of slaves is set at work for the master. The coordinator makes the identification of the slave known to the master by sending a reference of it to the master. With this information the master can activate the slave. Before the slave can really work, it should know on which grid (identified by the grid-of-grid coordinates n, m, l) it should perform the relaxation. The master has these coordinates available and writes them on its own output port. The coordinator takes care that the slave can read this information from its input port by setting up a stream between the output port of the master and its own input port. The master process continues its work and makes again a request for the creation of a slave process. When all the slaves are created and activated in this way, the master waits until the slaves are ready with the relaxation and are going to die. After this rendezvous, the master continues its sequential work until it again arrives at a point where it wants to use a pool of slaves to delegate the relaxations to.

In MANIFOLD we can easily realize the master/slave protocol described above in a general way in which the master and slave are parameters of the protocol. In this protocol we only describe how instances of the master and slave process definitions should communicate with each other. For the protocol it is irrelevant to know what kind of computations are performed in the master and slave. What is indeed important for the protocol is that the in/output and the event behavior of the master and slave are tuned to the protocol. E.g., the protocol manifold is only able to create a slave when the master requests for its creation by raising an event. Also, the master should write the data needed by the slave, on its own output port and the slave should read this information from its own input port, etc. For a stepwise description of the behavior interface of the master and slave manifold, and other details about the restructuring of the sequential code, we refer to [4]. In the next two sections we proceed by giving a speed-up analysis and speed-up results.

4.4 Speed-up analysis

All experiments have been run on a single multiprocessor machine in a real contemporary computing environment, i.e. an environment in which it cannot be guaranteed that one is the only user. In such an environment, care should be taken in interpreting speed-up numbers. This is shown in the following multi-user, single-machine analysis, in which we make these assumptions:

- the only processes which are significant with respect to the use of CPU time are computing processes,
- all computing processes get equal time slices from the scheduler of the machine and they totally consume these,
- the computational work embodied in a sequential program can be completely and equally distributed over parallel processes.

Then, with n the number of processors in a machine ($n \geq 1$), m_1 the number of processes our own application consists of ($m_1 \geq 1$; $m_1 = 1$ representing the sequential application) and m_2 the number of processes from other users ($m_2 \geq 0$), we can write as expression for the investment of CPU power p in our own application:

$$p = \begin{cases} n\frac{m_1}{m_1+m_2}, & \text{if } m_1 + m_2 > n \\ m_1, & \text{if } m_1 + m_2 \leq n \end{cases} . \qquad (1)$$

With the investment of CPU power inversely proportional to the elapsed computing times needed, from (1), expressions for various speed-up factors can be derived. As examples, we look at two of these factors.

The first speed-up factor relates the computing times of our parallel application run on a multiprocessor machine ($n > 1, m_1 > 1$), to those of the sequential version run on a single-processor from that machine ($n = 1, m_1 = 1$), both runs with the same number m_2 of other processes. For the corresponding speed-up factor, to be denoted by s_{n,m_1}, it follows

$$s_{n,m_1} \equiv \frac{p(n>1, m_1>1, m_2)}{p(n=1, m_1=1, m_2)} = \begin{cases} n(1+m_2)\frac{m_1}{m_1+m_2}, & \text{if } m_1+m_2 > n \\ m_1(1+m_2), & \text{if } m_1+m_2 \leq n \end{cases} . \quad (2)$$

Note that for the multi-user ($m_2 > 0$) situation, the speed-up s_{n,m_1} can be much larger than the number of processors n for the case $m_1 + m_2 > n$, and *is* always larger than the number of processes m_1 for the case $m_1 + m_2 \leq n$. In Figure 7, distributions of s_{n,m_1} are depicted for a 4-,8- and 16-processor machine, respectively. (Of course, the speed-up factors are defined at the integer points (m_1, m_2) only, the iso-lines as drawn in between these points are only meant to help in recognizing the discrete speed-up patterns.)

The second speed-up factor to be considered relates the computing times of our application when run on a machine with other processes running simultaneously, to those run on the same machine, but with no other processes. For the corresponding speed-up factor, to be denoted by s_{m_2}, it follows

$$s_{m_2} \equiv \frac{p(n, m_1, m_2=0)}{p(n, m_1, m_2>0)} = \begin{cases} \frac{m_1+m_2}{m_1}, & \text{if } m_1 > n \\ \frac{m_1+m_2}{n}, & \text{if } m_1 \leq n \text{ and } m_1+m_2 > n \\ 1, & \text{if } m_1+m_2 \leq n \end{cases} . \quad (3)$$

In Figure 8, distributions of s_{m_2} are depicted for a 4-,8- and 16-processor machine, respectively. Formula (3) may be practically relevant in comparative studies. With (3), from elapsed times measured in an environment in which a known number of other processes have been running simultaneously, one may approximately calculate the corresponding times in a hypothetical single-user ($m_2 = 0$) environment. With the theoretical s_{m_2} computed from (3) and with the real (elapsed) time $t_{\text{real}}(m_2 > 0)$ measured, we may estimate the corresponding elapsed time in single-user ($m_2 = 0$) environment by

$$t(m_2 = 0) = s_{m_2}\, t_{\text{real}}(m_2 > 0). \quad (4)$$

All our experiments have been done during quiet periods of the system ($m_2 \approx 0$). Therefore, since we mostly had $m_1 > n$, in our case $s_{m_2} \approx 1$ holds.

4.5 Speed-up results

All experiments have been run on an SGI Challenge L with four 200 MHZ IP19 processors, each with a MIPS R4400 processor chip as CPU and a MIPS R4010 floating point chip for FPU. This 32-bit machine has 256 megabytes of main memory, 16 kilobytes of instruction cache, 16 kilobytes of data cache, and 4 megabytes of secondary unified instruction/data cache. The machine runs under IRIX 5.3, is on a network, and is used as a server for computing and interactive jobs. Other SGI machines on this network function as file servers.

Computations have been done for both the sparse- and the semi-sparse grid approach. For the sparse-grid approach, the finest grid levels considered are: 1,2 and 3, for the semi-sparse-grid approach, these are: 2,4 and 6. (For both approaches, the dynamic creation of slaves in different work pools is shown in [4].)

The results of our performance measurements for the sparse- and the semi-sparse grid approach are given in Tables 2 and 3, respectively. They show the elapsed time versus the grid level. To even out such unpredictable effects as network traffic and file server delays, etc., we have run the two versions of the application on each of the three levels close to each other in real time, and for each version of the application, five times on each level. From Tables 2 and 3, it appears that the MANIFOLD version takes good advantage of the parallelism offered by the four processors of the machine. For the sparse-grid and the semi-sparse-grid application, the MANIFOLD-code times are about 3.25 and 3.75 times smaller, respectively, than the sequential-code times. So, in both cases we have obtained a nearly linear speed-up.

5 Conclusions

The intrinsically low computational complexity of sparse-grid and semi-sparse-grid techniques, plus the additional gains in computing time by parallelization, make both methods challenging for very computing-intensive work.

An interesting possibility for future research is the application of local 3D semi-refinement. For this, the relative truncation errors that are available between all grids on two consecutive grid levels, can serve in the grid-adaptation criterion. Work in this direction is in progress.

References

[1] ARBAB, F.: Coordination of massively concurrent activities, Report CS-R9565, CWI, Amsterdam (1995). Available on-line at http://www.cwi.nl/ftp/CWIreports/IS/CS-R9565.ps.Z.

[2] ARBAB, F.: The IWIM model for coordination of concurrent activities, in: *Coordination Languages and Models, Proceedings of Coordination '96* (CIANCARINI, P. AND HANKIN, C., EDS.), *Lecture Notes in Computer Science*, **1061**, 34–56, Springer, Berlin (1996).

[3] BRANDT, A.: Guide to multigrid development, in: *Multigrid Methods*, Proceedings, Köln-Porz, 1981 (HACKBUSCH, W. AND TROTTENBERG, U., EDS.), *Lecture Notes in Mathematics*, **960**, 220–312, Springer, Berlin (1982).

[4] EVERAARS, C.T.H. AND KOREN, B.: Using coordination to parallelize sparse-grid methods for 3D CFD problems, *CWI-Report*, CWI, Amsterdam (to appear).

[5] GRIEBEL, M., ZENGER, C. AND ZIMMER, S.: Multilevel Gauss-Seidel-algorithms for full and sparse grid problems, *Computing*, **50**, 127–148 (1993).

[6] HEMKER, P.W.: Sparse-grid finite-volume multigrid for 3D-problems, *Advances in Computational Mathematics*, **4**, 83–110 (1995).

[7] HEMKER, P.W. AND KOREN, B.: A non-linear multigrid method for the steady Euler equations, in: *Proceedings of the GAMM-Workshop on the Numerical Simulation of Compressible Euler Flows*, Rocquencourt, 1986 (DERVIEUX, A., LEER, B. VAN, PÉRIAUX, J. AND RIZZI, A., EDS.), *Notes on Numerical Fluid Mechanics*, **26**, 175–196 Vieweg, Braunschweig (1989).

[8] HEMKER, P.W. AND SPEKREIJSE, S.P.: Multiple grid and Osher's scheme for the efficient solution of the steady Euler equations, *Applied Numerical Mathematics*, **2**, 475–493 (1986).

[9] HEMKER, P.W. AND ZEEUW, P.M. DE: BASIS3, a data structure for 3-dimensional sparse grids, in: *Euler and Navier-Stokes Solvers Using Multi-Dimensional Upwind Schemes and Multigrid Acceleration* (DECONINCK, H. AND KOREN, B., EDS.), *Notes on Numerical Fluid Mechanics*, **57**, 443–484, Vieweg, Braunschweig (1997).

[10] KOREN, B.: Defect correction and multigrid for an efficient and accurate computation of airfoil flows, *Journal of Computational Physics*, **77**, 183–206 (1988).

[11] KOREN, B.: Multigrid and defect correction for the steady Navier-Stokes equations, *Journal of Computational Physics*, **87**, 25–46 (1990).

[12] KOREN, B. HEMKER, P.W. AND ZEEUW, P.M. DE: Semi-coarsening in three directions for Euler-flow computations in three dimensions, in: *Euler and Navier-Stokes Solvers Using Multi-Dimensional Upwind Schemes and Multigrid Acceleration* (DECONINCK, H. AND KOREN, B., EDS.), *Notes on Numerical Fluid Mechanics*, **57**, 547–567, Vieweg, Braunschweig (1997).

[13] MULDER, W.A.: A new multigrid approach to convection problems, *Journal of Computational Physics*, **83**, 303–323 (1989).

[14] NAIK, N.H. AND ROSENDALE, J. VAN: The improved robustness of multigrid elliptic solvers based on multiple semicoarsened grids, *SIAM Journal on Numerical Analysis*, **30**, 215–229 (1993).

[15] OSHER, S. AND SOLOMON, F.: Upwind difference schemes for hyperbolic systems of conservation laws, *Mathematics of Computation*, **38**, 339–374 (1982).

[16] RADESPIEL, R. AND SWANSON, R.C.: Progress with multigrid schemes for hypersonic flow problems, *Journal of Computational Physics*, **116**, 103–122 (1995).

[17] RÜDE, U.: Multilevel, extrapolation, and sparse grid methods, in: *Multigrid Methods IV*, (HEMKER, P.W. AND WESSELING, P.W., EDS.), Proceedings of the Fourth European Multigrid Conference, Amsterdam, 1993, *International Series of Numerical Mathematics*, **116**, 281–294, Birkhäuser, Basel (1994).

Table 1: Complexity and cost (of the ONERA M6 case) for the three types of grid-of-grids.

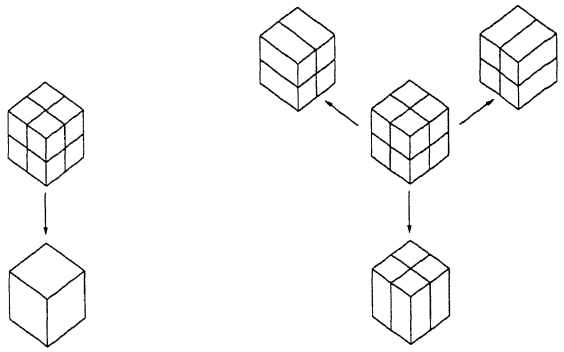
grid-of-grids method	full	semi-sparse	sparse
complexity	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2 \log^2 N)$	$\mathcal{O}(N \log^2 N)$
(scaled) CPU time	150	35	1

Table 2: The elapsed times (in hours:minutes:seconds) for the sparse-grid-of-grids approach.

	level	1*st* time	2*nd* time	3*rd* time	4*th* time	5*th* time
sequential	1	11.09	11.22	11.23	11.28	13.20
	2	1:35.54	1:35.87	1:36.56	1:39.82	1:41.22
	3	9:14.00	9:14.73	9:15.53	9:16.42	9:28.44
parallel	1	5.73	5.78	5.81	5.94	7.02
	2	33.19	33.25	34.11	34.82	35.58
	3	2:45.52	2:46.28	2:47.62	2:48.29	2:51.30

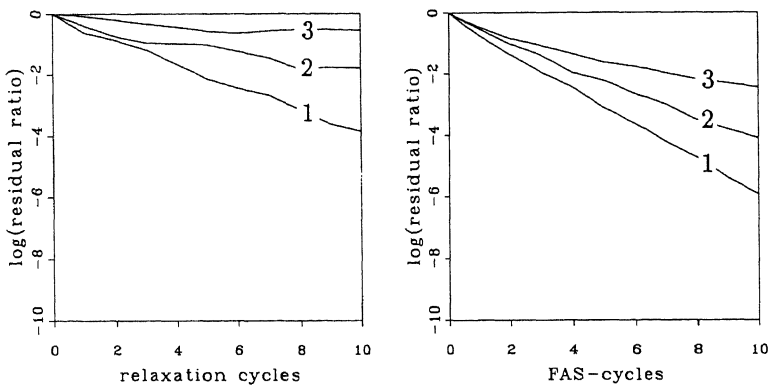
Table 3: The elapsed times (in hours:minutes:seconds) for the semi-sparse-grid-of-grids approach.

	level	1*st* time	2*nd* time	3*rd* time	4*th* time	5*th* time
sequential	2	50.07	50.26	50.47	50.55	52.20
	4	17:56.17	17:59.29	18:02.62	18:04.39	18:06.74
	6	4:33:03.59	4:33:07.66	4:37:07.13	4:38:10.83	4:51:59.52
parallel	2	26.47	26.50	27.70	27.80	28.48
	4	5:42.72	5:53.15	5:59.63	6:03.39	6:12.96
	6	1:13:34.67	1:13:54.51	1:15:04.86	1:15:12.74	1:23:22.07



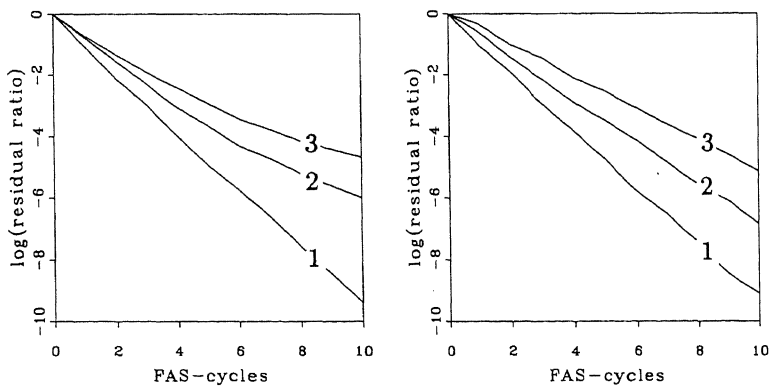
a. Full coarsening. b. Multiple semi-coarsening.

Figure 1: Two types of 3D coarsenings.



a. Single-grid. b. Standard multigrid.

Figure 2: Convergence behaviors of two solution methods, ONERA-M6 half-wing at $M_\infty = 0.84$, $\alpha = 3.06°$, $\Omega_{\lambda_{\max}} = (8 \times 2 \times 2) \times 2^{\lambda_{\max}}$-grid, $\lambda_{\max} = 1, 2, 3$.



a. With defect-dependent prolongation weights. b. With fixed prolongation weights.

Figure 3: Convergence behaviors of two semi-coarsened multigrid methods, ONERA-M6 half-wing at $M_\infty = 0.84$, $\alpha = 3.06°$, $\Omega_{l_{\max}, m_{\max}, n_{\max}} = (8 \times 2^{l_{\max}}) \times (2 \times 2^{m_{\max}}) \times (2 \times 2^{n_{\max}})$-grid, $l_{\max} = m_{\max} = n_{\max} = 1, 2, 3$.

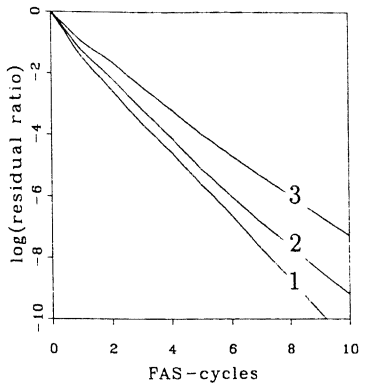
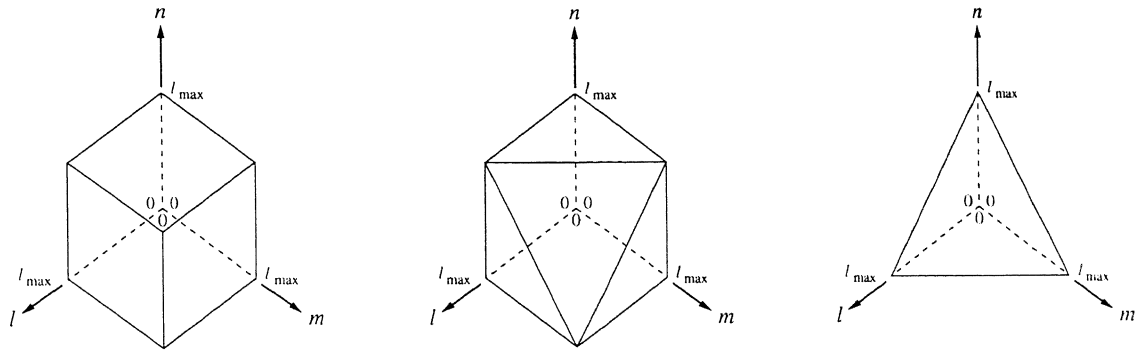
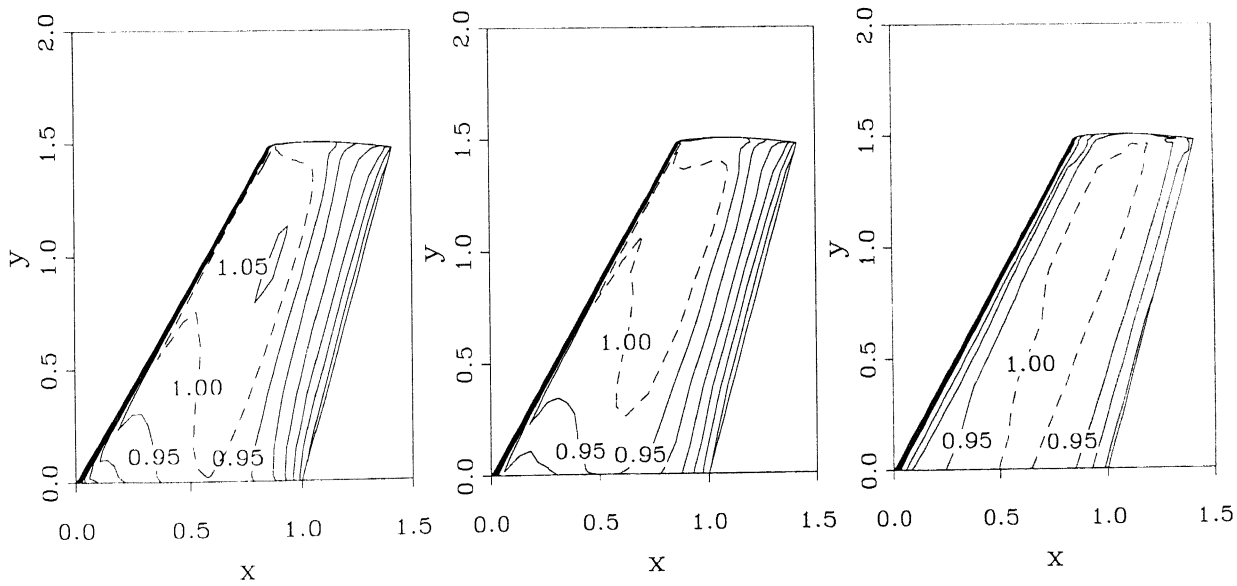


Figure 4: Convergence behavior of semi-coarsened multigrid method with fixed prolongation weights, V-cycles and level-by-level nested iteration, for ONERA-M6 half-wing at $M_\infty = 0.84$, $\alpha = 3.06°$, $\Omega_{l_{\max},m_{\max},n_{\max}} = (8 \times 2^{l_{\max}}) \times (2 \times 2^{m_{\max}}) \times (2 \times 2^{n_{\max}})$–grid, $l_{\max} = m_{\max} = n_{\max} = 1,2,3$.



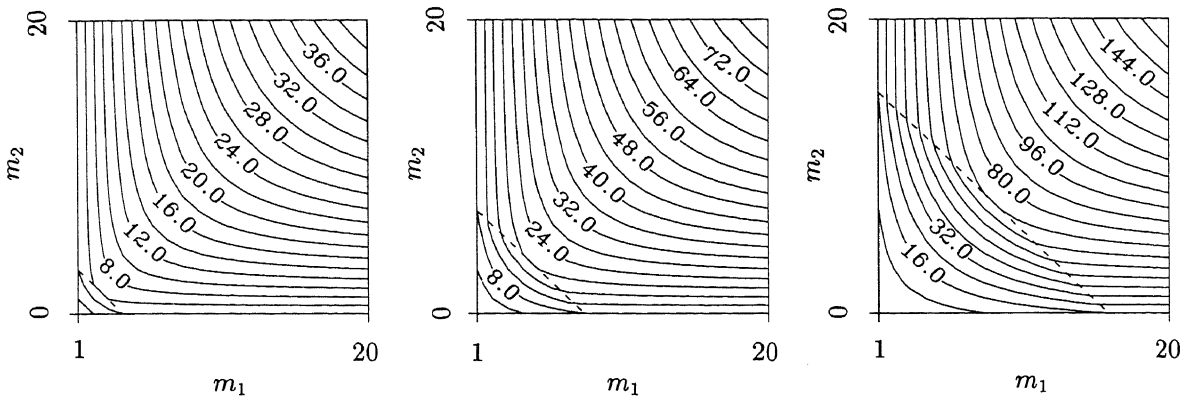
a. Full. b. Semi-sparse. c. Sparse.

Figure 5: Cubic, full grid-of-grids and the corresponding sparse and semi-sparse grid-of-grids.



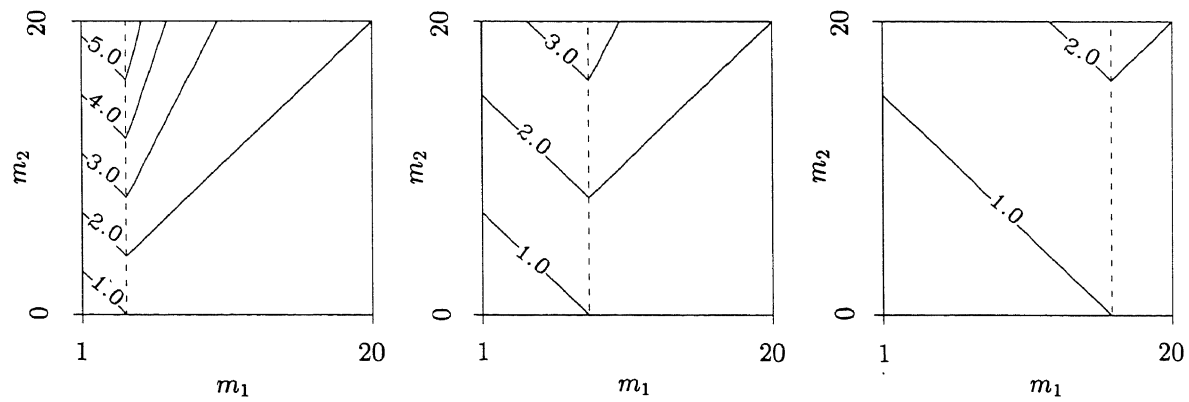
a. Full. b. Semi-sparse. c. Sparse.

Figure 6: Mach number distributions on upper half-wing surface for the three types of grid-of-grids.



a. $n = 4$. b. $n = 8$. c. $n = 16$.

Figure 7: Distributions of speed-up factors which can be expected when running a code containing m_1 parallel processes on a multi-processor machine ($n > 1$), instead of running the sequential version of that code on a single processor from that machine ($n = 1, m_1 = 1$), both applications with m_2 other processes running simultaneously.



a. $n = 4$. b. $n = 8$. c. $n = 16$.

Figure 8: Distributions of speed-up factors which can be expected when running a code containing m_1 parallel processes together with m_2 other processes ($m_2 > 0$), instead of with no other processes ($m_2 = 0$), both runs on an n-processor machine.